Integrazione di Sistemi Embedded
Laboratorio 04

Matteo Perotti 251453
Giuseppe Puletto 251437
Luca Romani 255244
Giuseppe Sarda 255648

November 24, 2018

1 Introduzione

Il quarto laboratorio ha come obiettivo l'esplorazione delle funzionalità che il comando **make** può offrire tra le quali il testing e la generazione automatizzata del file eseguibile. Strumento fondamentale per questa fase è il file **Makefile** usato da make.

2 Approccio alla scrittura del Makefile

Per la scrittura del Makefile, il quale contiene tutte le "ricette" necessarie per la completa compilazione di un progetto, si è consultato il manuale GNU relativo al comando *make* reperibile alla pagina web GNU make.
Di seguito sono riportate le informazioni più rilevanti ricavate dal manuale, alcune delle quali sono state usate all'interno dei Makefile prodotti.

Phony targets

Usando il "token" **.PHONY** si riesce ad indicare un target che non corrisponde al nome di un vero e proprio file, ma ad una ricetta eseguita soltanto se espressamente richiesto. Questa direttiva è utile in caso di omonimia tra un target e un file, i quali però non sono logicamente collegati, e che quindi, senza un'indicazione specifica, sarebbero erroneamente vincolati. Dichiarando dunque:

```
.PHONY: clean
clean:
        #do something
```

il target *clean* viene eseguito esclusivamente quando richiesto all'esecuzione del comando make evitando eventuali malfunzionamenti nel caso esita un file nominato "clean" nella directory del progetto.

Versioni del Makefile

Il Makefile è stato sviluppato in diversi modi, utilizzando regole esplicite ed implicite e variando il compilatore e le opzioni di compilazione in modo statico oppure dando la possibilità di scelta all'utente.

Versione 1 In questa versione del makefile è stata definita una variabile "CC" per rendere più pulito il file che di default imposta come compilatore gcc.
Il make file ha un target fittizio "all" senza ricette e con un'unica dipendenza: il file eseguibile che si vuole creare, universalDrawer in questo caso. In questo modo il makefile cerca subito quali sono i target che servono per la dipendenza dell'eseguibile.
Volendo strutturare il file in modo esplicito, è stato definito un target per ogni file oggetto utile a generare l'eseguibile. Tra le dipendenze di ogni file oggetto sono stati inseriti i file .c e .h necessari per la compilazione. Come ultimo target è stato definito "clean" che è senza dipendenze e che semplicemente rimuove, se richiesto, tutti i file oggetto e il file eseguibile presenti nella stessa directory del makefile

Versione 2

Versione 3 Il terzo makefile assolve le specifiche richieste mediante l'uso di variabili e compilazione condizionale. Dapprima si controlla se la variabile PREFIX è stata definita all'esecuzione del comando make: se sì, viene utlizzato come compilatore $(PREFIX)-gcc, in caso contrario gcc. La definizione o meno di PREFIX condiziona il valore della variabile OPZIONE. Se si definisce PREFIX=arm-none-eabi, viene eseguito il compilatore arm-none-eabi con la specifica addizionale –specs=nosys.specs. Tutto questo è possibile mediante l'uso di un'ulteriore compilazione condizionale che fa uso delle variabili COMPILATORE_1 e COMPILATORE_2. Qualunque sia il compilatore scelto, la compilazione viene eseguita sempre con l'opzione -static. Il Makefile si serve anche di altre tre variabili, al fine di specificare il percorso dei file.c, e le cartelle di destinazione dei file oggetto ed eseguibile. Quest'ultime vengono create rispettivamente attraverso la ricetta del target main.o e del target universalDrawer. Il target clean rimuove ricorsivamente le cartelle

creata e i file oggetto ed eseguibile al loro interno. Anche in questo Makefile si fa uso del target fittizio all.

Versione 4 Il Makefile è stato fatto per sfruttare il più possibile le regole implicite. Tutti i files .c vengono tradotti nei corrispettivi files oggetto .o.

Versione 5 Il Makefile ha le stesse funzionalità previste nelle altre versioni ma in questo caso presenta come target aggiuntivo "test", il quale ha come dipendenza l'eseguibile del programma e come ricetta l'esecuzione dello script "test_lab3_script.sh". Lo script serve per verificare che il programma universalDrawer funzioni correttamente confrontando il suo output con quello di un file di test.

3 Implementazione del test

Per la realizzazione del target test si è deciso di dividere la verifica del codice scritto per il precedente laboratorio in 4 parti:

- Creazione di uno script Python in grado di generare comandi di disegno casuali sotto specifiche
- Scrittura di un codice Python, basato sugli identici algoritmi usati durante il precedente lab, che si comportasse secondo specifiche relative alla parte funzionale ma che lasciasse totale libertà implementativa
- Esecuzione dei comandi generati al punto uno da parte del codice C e dello script Python
- Confronto dell'output finale delle due esecuzioni

Si noti che i limiti di questo test stanno nel fatto che, nel caso in cui gli algoritmi di disegno fossero sbagliati in partenza, non c'è modo di accorgersene. I casi limite inoltre non vengono così testati perché i comandi al punto 1 vengono per definizione prodotti al fine di rispettare le specifiche.

Generazione dei comandi

Lo script per la generazione dei comandi accetta dall'utente un certo numero di comandi che vengono creati usando la funzione *randint* contenuta nel modulo *random*. Infine i comandi generati vengono scritti all'interno di un file che verrà poi passato ai file eseguibile. L'esecuzione finisce con la scrittura di un **token** che segnala la fine dello stream di comandi.

Script di test

Lo script di test viene eseguito mediante il comando "make test" (usando la versione 5 del Makefile). Viene prima lanciato il programma in python inputGenerator.py per generare i comandi di prova. Questi, contenuti nel file cmd.txt, vengono letti riga per riga e salvati dentro una variabile di appoggio. La sequenza dei comandi di prova viene passata attraverso una pipe all'eseguibile, che, opportunamente modificato, crea un file di uscita con la matrice modificata. Viene infine eseguito il file universalDrawer.py, che produce il file py_matrix.txt, contenente l'output che ci si aspetta dall'eseguibile. Il file con la matrice appena generata e il file con la matrice attesa py_matrix.txt vengono confrontati con il comando "diff -s" per segnalare la presenza o meno di errori.

Modifica del programma

Il programma è stato modificato per poter riconoscere il comando di QUITTING ('Q'). Appena esso viene ricevuto viene chiamata una funzione che apre un file, disegna la matrice (frameBuffer) e poi lo chiude. In questo modo diventa semplice il confronto tra le due matrici generate dai programmi scritti in diversi linguaggi. L'interfaccia del programma è stata anche regolarizzata per rientrare nelle specifiche di interfaccia.